THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ELECTRICAL ENGINEERING

A MULTI-SENSOR EMBEDDED MICROCONTROLLER SYSTEM FOR CONDITION MONITORING OF RC HELICOPTERS

Artit C. Jirapatnakul

Spring 2005

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Electrical Engineering
with honors in Electrical Engineering

Approved: _____ Date: _____
Amulya Garga
Thesis Supervisor

_____ _____
Charles Croskey
Thesis Supervisor

_____ _____
Charles Croskey
Honors Adviser

_____ _____
Salvatore Riggio
Faculty Reader

ABSTRACT

Helicopters are complex, high performance machines designed to ensure the safety of their occupants during their expected lifetimes. To accomplish their goals, helicopters require extensive maintenance during their lifetimes at set intervals, whether necessary or not. To help alleviate the need for unnecessary maintenance, condition-based maintenance systems are under heavy development, with the military expressing much interest in such systems. As the name implies, condition-based maintenance systems rely on information about the condition of various mechanical components to determine when maintenance is necessary. This has the potential to greatly reduce cost and enhance safety.

The system developed here uses data from three sensors to monitor the condition of a radio-controlled helicopter. Data from the sensors is transmitted to a microcontroller, where it is processed before being transferred to other microcontrollers and eventually to a computer for storage and visualization. The system must fulfill several requirements imposed by the constraints of the radio-controlled helicopter, mainly small size and low power usage, while still being powerful enough to perform processing on sensor data with a bandwidth on the order of a kilohertz.

The system is able to communicate with three different types of sensors, an inertial measurement unit, global positioning satellite receiver, and a digital compass, perform calculations of statistics, and transmit the processed data reliably to a computer, where it is displayed in a custom graphical interface. The system is able to reliably handle

data from the highest bandwidth sensor available, the inertial measurement unit with a bandwidth of approximately 400Hz. Based on these results, the system is expected to be able to handle up to data up to a kiloHertz in bandwidth; higher data rates will require additional hardware. Thus, this system is suitable for the processing of moderate bandwidth sensor data and fulfills the design goals of the system.

TABLE OF CONTENTS

ABSTRACT ii

TABLE OF CONTENTS iv

LIST OF FIGURES vi

LIST OF TABLES vii

ACKNOWLEDGEMENTS viii

Chapter 1 Introduction 1

1.1 Background 1
1.2 Project Overview 2

Chapter 2 System Design and Implementation 4

2.1 Functional overview 4
2.2 Materials 6
2.3 Microcontroller and hardware description 8
2.4 Communication protocols 10
2.4.1 Microcontroller to computer communication protocol 10
2.5 Microcontroller program description 11
2.6 VB.NET program description 14

Chapter 3 Results and Conclusion 16

3.1 Results 16
3.1.1 PIC18 Results: Testing with sensors 16
3.9.2 PIC18 Results: Statistics with known data 18
3.9.3 dsPIC Results: Performing RMS calculation on IMU data 20
3.9.4 dsPIC Results: Analog-to-digital converter 22
3.10 Discussion 23
3.11 Conclusion 25

References 28

Appendix A List of Materials 29

A.1 Hardware 29

A.2 Software and development tools 30

Appendix B Processor Specifications 31

B.1 PIC18F8720 31
B.2 dsPIC30F6014 32

Appendix C Sensor Communication 33

C.1 IMU: Crossbow IMU400CC 33
C.2 GPS: Novatel Superstar II 35
C.3 Compass: Honeywell HMR3100 37

Appendix D Dallas Semiconductor One-wire Cyclic Redundancy Check 38

Appendix E Schematics 40

E.1 Additional RS-232 ports 40

Appendix F Academic Vita of Artit Jirapatnakul 41

LIST OF FIGURES

Figure 1: Sensors mounted in plastic box (from left to right: IMU, GPS, Compass)..6

Figure 2: System connections using two PIC18 development boards.....9

Figure 3: Packet structure for microcontroller protocol 11

Figure 4: Data obtained from PIC18 boards with IMU, GPS, and Compass Sensors.....18

Figure 5: Percent error in PIC18 RMS calculation as compared to MATLAB simulation 19

Figure 6: Percent error in PIC18 skew calculation compared to MATLAB simulation 19

Figure 7: Percent error in PIC18 kurtosis calculation compared to MATLAB simulation 20

Figure 8: RMS and unprocessed values from dsPIC board connected to the IMU 21

Figure 9: PIC18F8720 specifications.....31

Figure 10: dsPIC30F6014 specifications.....32

Figure 11: Schematic of additional RS-232 ports.....40

LIST OF TABLES

Table 1: Speed of various math operations on the dsPIC30F6014 @ 30MHz 24

Table 2: Bill of materials 29

Table 3: List of software and development tools 30

Table 4: IMU Data Packet Format 34

Table 5: GPS receiver packet format 36

Table 6: Compass packet format 37

ACKNOWLEDGEMENTS

I would like to thank Dr. Amulya Garga for his guidance and support in my work on this system. He was an immense help at all stages of my research. Robert Ceschini shared his vast knowledgeable about the PIC microcontroller as well with general hardware with me, helping to decrease my initial learning curve. I would also like to acknowledge the aid of Michael McArtor, another student in electrical engineering who worked with me on the system during the summer and helped in programming for the PIC18 microcontroller and the graphical interface. Additional thanks go to Dr. Joseph Horn and Hsun-Hsuan Huang who provided me with information on the sensors as well as the sensors themselves. Finally, I would like to thank my honors advisor, Dr. Charles Croskey, for guiding me through my undergraduate career, and Dr. Salvatore Riggio for reading my thesis. Funding for this project was graciously provided by the Applied Research Lab at Pennsylvania State University.

Chapter 1

Introduction

1.1 Background

Helicopters are designed to provide high performance and safety within their lifetimes. However, helicopters are complex machines with many different interacting systems that make maintenance costly and difficult. Compounding the problem is a tendency for helicopters to be used in service past their expected lifetimes and in conditions that were not conceivable many years ago. This situation encourages the use of monitoring systems to preserve the current performance of helicopters and to enhance safety. These monitoring systems are useful not just for older helicopters, but for new ones as well, where additional information about the condition of the helicopter can aid the crew in making decisions regarding whether to undertake risky missions. With these benefits in mind, the U.S. Army is currently developing health-and-usage monitoring systems (HUMS) for its new UH-60M Black Hawk and Block III AH-64D Apache helicopters[1]. Not only can HUMS improve safety and reliability, but they are also important in reducing operating costs. Many different methods and algorithms exist to extract key indicators from different types of sensor data to both monitor health and predict the failure of mechanical components. Future work in this area is expected to improve the accuracy of HUMS.

Although health-and-usage monitoring systems already exist, with several commercial implementations, this is still a new field. Research on novel algorithms to more effectively isolate abnormal data continues; yet, there is no easy way to determine if the algorithms will work when actually implemented – testing with simulated data can only go so far. A platform for testing new algorithms would help in determining the viability of new algorithms.

1.2 Project Overview

This project involves the development of condition monitoring system for a radio-controlled (RC) helicopter – it is difficult to obtain access to real helicopters for testing and implementation on an RC helicopter can still yield useful information. The basic requirements for such a system are: communicating with multiple different sensors, performing processing on the data obtained from the sensors, and finally transmitting the results to a computer for storage and display. The desire to have a practical system, as well as the use of an RC helicopter, imposed the following additional constraints on this system: 1) low cost, 2) use of commercial hardware, 3) compact size, and 4) low-power consumption. The first two goals are interrelated; generally, standard components are available at much lower cost than proprietary devices. Further, the use of commercial hardware insures availability of parts and a large knowledge base. The next goal, compact size, is due to the space limitations of a small RC helicopter. In order to mount the system in the helicopter for testing, the system must be as compact as possible. The lack of space in the helicopter plays a role in the final constraint, low power

consumption. Large batteries can not be used due to limitations in the lift capacity of the radio-controlled helicopter; yet the system needs to be able to run for at least one flight. To meet this requirement, the system needs to restrict its current draw to the order of milliamps. All these project goals severely limited the selection of suitable hardware for this project.

In consideration of these goals, the system was designed to use many smaller processing boards instead of one large, high-performance processing unit. This offers the possibility of reducing the overall power consumption and cost of the system through the use of inexpensive, low-power microcontrollers in the system. Since the processing requirements for different types of sensors vary; for example, an accelerometer with a high sampling rate would need considerably more processing capability than a temperature sensor sampling data every second, the processing capabilities can be customized for each sensor, making the system as power-efficient as possible.

The PIC18 and dsPIC families of microcontrollers were chosen for the project. These families offer several low-cost/low-power chips with a large amount of RAM and a high clock rate, relative to microcontrollers in general. The PIC18 is typical microcontroller suited for non-math intensive operations with many general-purpose input/output pins. The dsPIC includes a DSP core in addition to a normal microcontroller core on chip, making it more appropriate for complex processing. The PIC18 microcontrollers are intended for use with low data rate sensors, such as temperature, while the dsPIC is primarily for high data rate sensors, such as an accelerometer, and for performing more math-intensive signal processing.

Chapter 2

System Design and Implementation

2.1 Functional overview

The functionality required of this system for condition monitoring on an RC helicopter can be divided into three tasks: communication with the sensors, processing of the data, and communication with a computer.

The system is designed to interface with sensors either using the RS-232 serial interface or the analog-to-digital converters (ADC) on-board the microcontrollers. The RS-232 interface was chosen to ensure compatibility with the largest possible number of sensor devices. Additionally, the use of a digital interface relieves some processing load that comes from processing data from the ADC. All three sensors used in the design and testing of this system have RS-232 interfaces. Although the RS-232 interface has many advantages, it is generally limited to a maximum data rate of 115.2 kilobits per second (kbps), though most devices use much lower speeds. For applications requiring higher data rates, the ADC on-board the microcontrollers can be used. The PIC18F8720 has a 10-bit ADC capable of fast sampling rates in the kilohertz range (the actual sampling rate depends on the external circuit), which may not be much better than the bandwidth offered by the digital interfaces, but the dsPIC30F6014 has a 12-bit ADC capable of sampling up to 100kHz.

For the sensors using the RS232 interface, additional processing must be done by the microcontroller before data can be obtained. Each sensor uses a different protocol to encapsulate its data, which is briefly discussed in Section 2.4. The protocol is necessary to enable the recipient of the data to determine where a packet starts and ends; there is no hardware synchronization to indicate the start/end of data. Once the protocol is decoded and the data extracted, the microcontroller might have to perform additional processing, such as reversing the byte order or concatenating different bytes together, to form the correct numeric representation in the microcontroller's memory. Data obtained from the ADC does not require this additional processing.

Once the sensor data is available in a useful format, further data processing can be performed by the microcontrollers. This may range from simple statistical analysis, such as computing root-mean-squared values for the incoming data, to more complex frequency-domain analysis. The primary goal in performing this processing is to reduce the huge volume of data from the many different sensors to only the features of interest.

After the necessary computations are performed, the processed data is transferred to a board with a microcontroller whose primary purpose is communication with the PC. This board multiplexes the data obtained from the various other boards and sensors into one data stream for transmission to a PC for visualization and data storage. This requires encapsulating the packets in a custom protocol described in Section 2.4.1.

The encapsulated data is transferred to the computer using the RS232 protocol. Running on the computer is a program written in VB.NET that decodes the custom protocol and displays the data textually and graphically, as well as logging the data to different files.

2.2 Materials

This system requires several hardware components and software programs. The key hardware components of this system are the microcontrollers and the sensors. This system uses three different sensors: an inertial measurement unit (IMU), a global positioning system receiver (GPS), and a compass. The IMU in this system is a Crossbow IMU400CC that is capable of measuring linear acceleration along three orthogonal axes and rotation rates around three orthogonal axes. This sensor provides the majority of the data useful for health monitoring of the RC helicopter. A Novatel Superstar II is the GPS receiver for the system, providing data about the location and speed of the helicopter. The current heading of the helicopter is determined by a Honeywell HMR3100 digital compass. These three sensors are mounted in a plastic box as depicted in Figure 1 and are powered by 2 9.6V nickel-metal hydride (NiMH) rechargeable battery packs and a 9V alkaline battery.

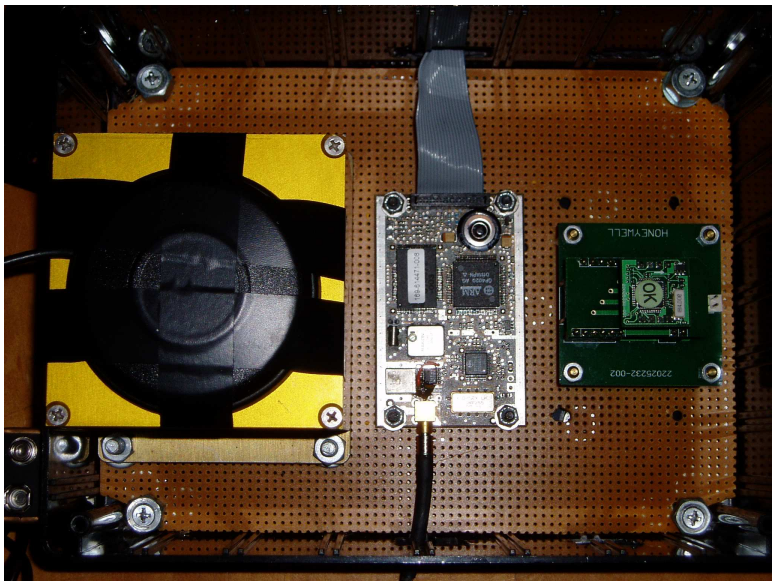

Figure 1: Sensors mounted in plastic box (from left to right: IMU, GPS, Compass)

To communicate with, and process data from the aforementioned sensors, microcontrollers from two different families made by Microchip are used in this system, a PIC18F8720 and a dsPIC30F6014. The PIC18F8720 is a 16-bit RISC processor capable of executing 10 million instructions per second (MIPS) with 3840 bytes of RAM [2]; additional specifications are located in Appendix B.1. This microcontroller was chosen for its low cost and power consumption, as well as relative ease of programming. Two of these microcontrollers were used in the system for communicating with the three sensors. The dsPIC30F6014 is a 16-bit RISC CPU with support for digital signal processing (DSP) functions capable of executing 30 MIPS with 8192 bytes of RAM [3]; additional specifications are located in Appendix B.2. The dsPIC is extremely inexpensive and consumes little power for a controller with DSP functionality; the additional capability of the dsPIC is used to process the data from the IMU sensor.

For ease of use, the microcontrollers were purchased already mounted on development boards. The development board for the PIC18 offered all supporting circuitry for the microcontroller as well as an assortment of LEDs, an RS232 port, and an in-circuit debugger/programming interface. The board also offered a small development area for constructing additional circuitry. The development board for the dsPIC offered the capabilities above along with an additional RS232 port, character LCD, and audio input/output. With these development boards, work on programming could begin quickly.

The microcontrollers are programmed using embedded C. For the PIC18F8720, a development environment from CCS called PICC is used. The C compiler supports most of the ANSI C specification and is capable of generating compact, optimized code. Unfortunately, this compiler is not yet available for the dsPIC30F6014; for the dsPIC,

Microchip's own C30 compiler is used. This also supports most of the ANSI C specification, but has libraries that differ in usage from the libraries available with the PICC compiler, resulting in an additional learning curve. The resulting compiled programs are downloaded to the processors via the programming interfaces on the development boards.

The components and software listed here do not encompass all the necessary materials, a full list of the hardware materials is included in Appendix A.1 and a full list of development tools is included in Appendix A.2.

2.3 Microcontroller and hardware description

The hardware was based around development boards from Microchip. These boards, described above, provided all supporting circuits to the chip, including power and oscillator connections, as well as conveniently providing headers to make connections to the pins on the microcontroller. ThePIC18 development board had only one RS-232 port; two additional ports were added to enable two sensor devices to be connected to the board simultaneously. The PIC18 has two UART modules on-chip – one is used by the original RS-232 port on the board, the other was used to control an additional port. A dual RS-232 driver chip, a Maxim MAX232, is used to convert the voltage to the proper levels for RS-232. For the third port, another UART was necessary. This was accomplished through the use of a Maxim MAX3100, an SPI compatible UART. The MAX3100 is connected to the SPI module on the PIC18, and its outputs are connected to the second set of inputs on the MAX232 to provide the correct voltage to the RS-232

port. A schematic of these additional ports is included in Appendix E.1. Since the dsPICDEM development included two RS-232 ports, no additional ports needed to be added to it.

As previously mentioned, the system is comprised of several development boards. This is done to distribute processing as well as provide enough ports to connect to all the sensors. A diagram of the connections between the boards and sensors, as well as the speed of the links, is shown in Figure 2. This is only a representative connection diagram; the boards and sensors can be moved around depending on the requirements of the particular application.

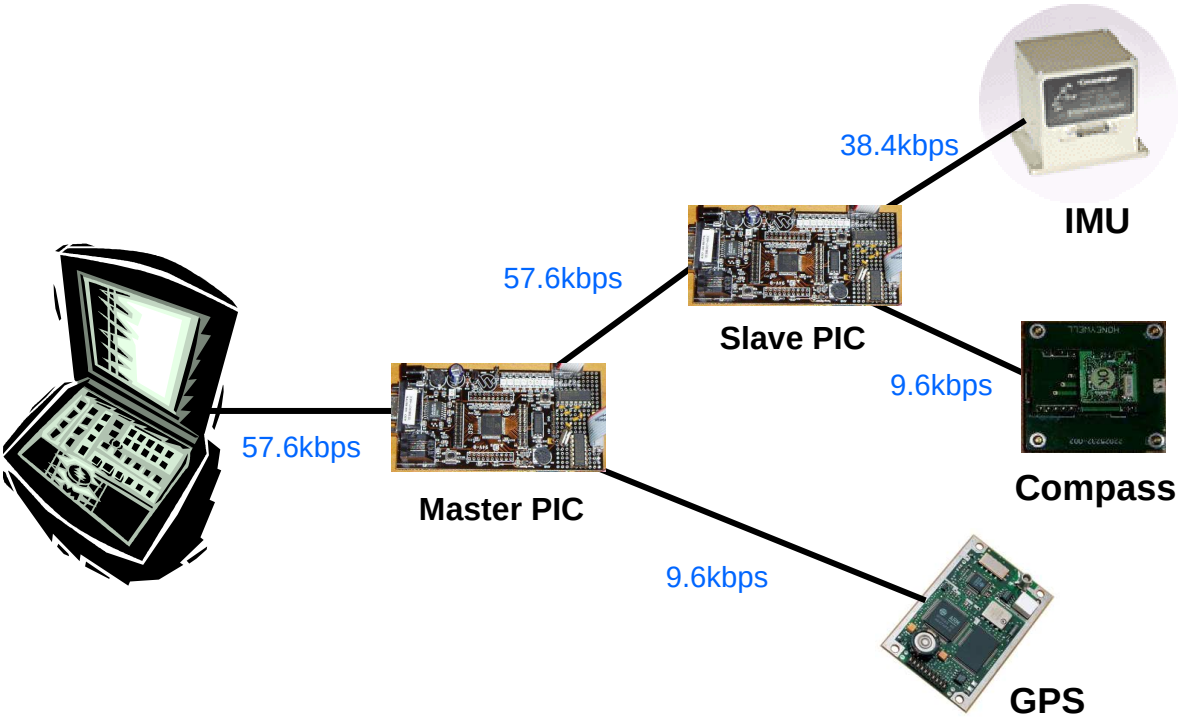

Figure 2: System connections using two PIC18 development boards

2.4 Communication protocols

To understand some of the programming involved in this system, the basics of the communications protocols used by the sensors and the microcontroller must be introduced. In general, data to be transmitted is first encapsulated in a packet by preceding the data with header bytes and following the data with a checksum to ensure correct transmission. More details of the sensors' protocols are included in Appendix C, while a detailed description of the protocol between the microcontroller and the computer is included here due to its relevance to the development of the system.

2.4.1 Microcontroller to computer communication protocol

The protocol used to transfer data between the microcontroller and the computer is designed to be extensible and robust. This requires support for many different types of devices (e.g. GPS versus compass) as well as several instances of a given type of device. In addition, each sensor might have different types of messages to relay, for example, a GPS sensor might have a different message for position and its speed. Furthermore, the protocol must ensure that data arrives without errors. Ideally, the protocol would be able to recover from errors, but in this implementation, it is sufficient to be able to reliably determine when errors have occurred. Taking all these requirements into consideration, the following protocol was developed. A graphical representation of this protocol is depicted in Figure 3.

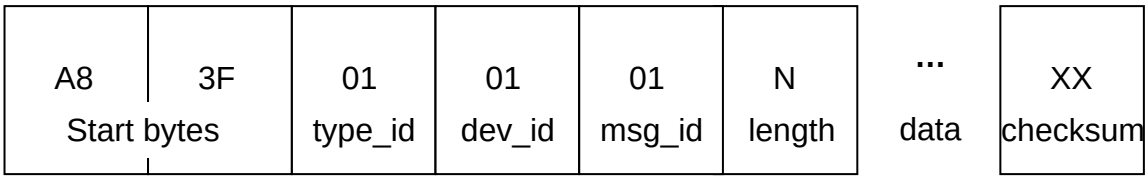

Figure 3: Packet structure for microcontroller protocol

The leftmost bytes in the figure are the first bytes to be transferred. Two bytes are used to indicate the start of a packet; a sequence of two bytes is used to help prevent false detection of the start of a packet. The next three bytes are used to identify the origin of the data. The type_id byte represents the type of device, the dev_id byte holds the serial number of the device, and the msg_id byte indicates the type of message. For these three bytes, the numbers 0 and 255 are reserved for the transmission of error flags. Thus, the protocol can accommodate up to 254 different types of devices, 254 instances of a particular type of device, and 254 different messages. This should provide ample capacity to expand the number of sensors used in this system. The length of the data follows next, which is included to aid in the decoding of the packet. After the header, the data is transmitted, followed by the checksum byte. The checksum used in this protocol is calculated using the checksum algorithm from Dallas Semiconductor's One-wire protocol, described further in Appendix D.

2.5 Microcontroller program description

The program developed for each microcontroller is divided into different modules: interfacing with the universal synchronous asynchronous receiver transmitter (USART), sensor packet decoding, data packet encoding, and data processing. The C

language is used, both for portability and future maintainability. The program logic is similar in the programs written for the PIC18 and dsPIC, so the following description is applicable to programs for both.

The low-level details of the RS-232 protocol are handled by the USART module built into the PIC18; interfacing with the USART module is accomplished by reading/writing data to special registers through library functions in the CCS C compiler and C30 compiler and using interrupts. The USART module has interrupts for both transmission and reception; when the transmit buffer is empty, the interrupt flag is set, indicating it is ready for more data. An interrupt service routine (ISR) reads data from a buffer array into the USART transmission buffer. Likewise, when the receive buffer is full, the receive interrupt flag is set, signifying data is ready for reading. Upon receiving an interrupt, the ISR reads data from the receive buffer into an array for later use in the main program. Access to these arrays is encapsulated by functions; in the main program, writing to the port is accomplished with a call to the WriteOutQue() function, and reading data is performed with a call to the ReadInQue() function. Both these functions interact with the data buffering arrays, allowing data to be buffered until the USART module is free to send or receive.

The PIC18 boards have a total of three serial ports, but the PIC18 only has two USART controllers on-board. The third serial port is implemented using a Maxim MAX3100 UART controller that communicates with the PIC18 using the serial peripheral interface (SPI) bus. The PIC18 includes an SPI controller which is easily accessible through libraries offered by the CCS C compiler. Additional functions were developed to communicate with the MAX3100, and these functions were integrated into

the WriteOutQue() and ReadInQue() functions above, to make the use of the MAX3100 transparent to the main program.

The next logical step in the program is to decode each sensors' protocol to extract data. For all the sensors, a finite state machine is used to keep track of the location of the current byte in the packet, with each incoming byte incrementing the state of the machine. Once the final state of the machine is reached, depending on the sensor, the checksum is verified, and the data is copied into the appropriate variables. For the IMU and GPS sensors, the data is stored in C structures that closely mimic the data ordering from the sensors.

Once the data is formatted appropriately for the microcontroller, the data can be processed using any type of algorithm that can fit within the constraints of each microcontroller. The algorithm is subject to change depending on the application; additional discussion is located in Section 3.1.

After the data is processed to reduce the volume of samples and to extract interesting features, it must be transferred to a computer. This is accomplished by packaging the data according to the protocol discussed in Section 2.4.1. A function was developed to perform this operation, send_data(). It takes the type id, device id, message id, and data length as parameters. The function calculates a checksum for the data and header bytes using the checksum algorithm in Dallas Semiconductor's 1-wire protocol, detailed in Appendix D. Finally, the function transmits the data packet over a serial connection to a computer at 115.2kbps.

2.6 VB.NET program description

On the computer, another program developed for this system waits for data. This program acts a visual interface for the received data, displaying the data as both text and graphically, if appropriate. The program was written in Visual Basic.NET (VB.NET) in order to rapidly prototype a working graphical interface. The operation of the program can be divided into two main parts, communicating with the microcontroller and displaying the data.

The communication with the microcontroller was analogous to communicating with the sensors – the computer must decode the packets in a similar manner. The first step is communicating with the serial port in VB.NET. Unfortunately, although previous versions of Visual Basic had included support for interfacing with the serial port, VB.NET had no such capability. To solve this problem, an open serial port class[1] by Corrado Cavalli was adapted for this system. This serial class uses an event-driven interface, which allows a function to be called in response to an event. This mechanism is used to read data from the serial port when data is available.

To decode the packets from the microcontroller, the program looks for the two start bytes specified by the protocol in Section 2.4.1. Once the start bytes are found, the program makes a new instance of a “Packet” class to store the incoming data. The next three bytes are stored in appropriate variables in the packet class along with the length. The program then reads the number of bytes indicated by the length and finally the checksum. The checksum for the data is calculated in the program using the same

[1] Available at http://www.codeworks.it/net/VBNetRs232.htm

algorithm as discussed in the previous section, and if the calculated checksum matches the received checksum, the packet is marked as complete.

After the program receives a complete packet, the program looks at the type id field to what type of data the packet contains. Based on this, it creates a new instance of a class appropriate to the packet, for example, an instance of the “IMU” class for IMU data. These classes were written to convert the raw data into VB.NET data types for easy display. The variables of these classes are used in the main program loop to populate text fields and add points to the graph. The graphing operations are handled by a third-party graph class from Dundas Software, Dundas Chart; a free version was included with the VB.NET resource kit. If logging of the data is desired, the program writes the incoming data into a comma-delimited file using file classes provided by the VB.NET environment.

The program works for its specified purpose, as will be later shown in the results section.

Chapter 3

Results and Conclusion

3.1 Results

To determine if the functionality described in Chapter 2 was correctly implemented, several incremental tests were performed. Not all tests produced interesting data, for example, initial tests of the serial port consisted of simply echoing received characters. Included in this discussion are tests used to verify correct operation of the entire system.

3.1.1 PIC18 Results: Testing with sensors

In this test, two PIC18 boards were connected together as depicted in Figure 2 in order to provide enough serial ports to connect all three sensors, due to each board only having two serial ports for connecting with sensors. The compass and GPS were connected to one board, designated as “slave.” The slave board was daisy chained to one serial port on another board, the “master”, and the other port was connected to the IMU. The “slave” board decoded the packets from the compass and GPS sensors and encapsulated the data using the microcontroller protocol described in Section 2.4.1. The master board decoded the data from both the master board and the IMU sensor, read from the temperature sensor on-board the development board, encapsulated them in the protocol again, and transmitted the data to the computer. Once on the computer, the data was processed for

display by the VB.NET program. The results are displayed in Figure 4. The data is correct, within a small margin of error. The location of State College, PA, as listed by the U.S. Census Bureau, is 40.792522 N, 77.852279 W[4], which corresponds with the latitude and longitude reported by the GPS. The IMU data is more difficult to judge, but, sitting still, the linear accelerations along the x and y axes are nearly 0 G, while the acceleration along the z-axis is almost 1 G, which is correct. The rotational acceleration values are also very low, all less than 0.5 degrees; this might have been a result of sensor noise. The compass heading changed correctly relative to the motion of the compass (i.e. rotating the sensor 360 degrees caused the reading to change and eventually return to the starting heading). The temperature also corresponded to the ambient room temperature at the time. These results indicate that packet processing and transmission/reception work correctly in the microcontroller programs and that the VB.NET is capable of correctly interpreting the data and displaying the results.

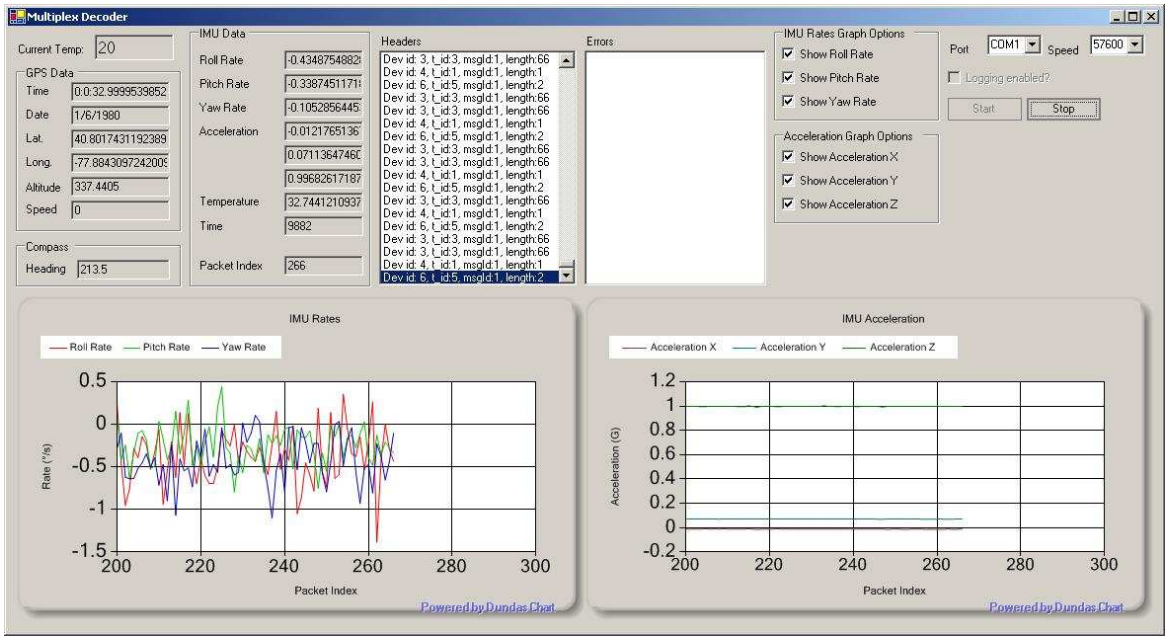

Figure 4: Data obtained from PIC18 boards with IMU, GPS, and Compass Sensors

3.1.2 PIC18 Results: Statistics with known data

In the first test, known data, packaged in a similar protocol format as one of the sensors, was transmitted to the PIC18 board. The PIC18 board calculated simple statistics – RMS, skew, and kurtosis. A sliding 10 sample window was used. The errors between the PIC18 RMS, skew, and kurtosis calculations and the same calculation performed by MATLAB are shown in Figures 5, 6, and 7. The error in all the calculations is on the order of 10^{-3} percent, an insignificant error especially considering the sensors in this system are not capable of such precision. The error is most likely caused by rounding caused by the limited precision of the PIC18 processor; the random distributions of points in the skew and kurtosis error graphs support this assertion.

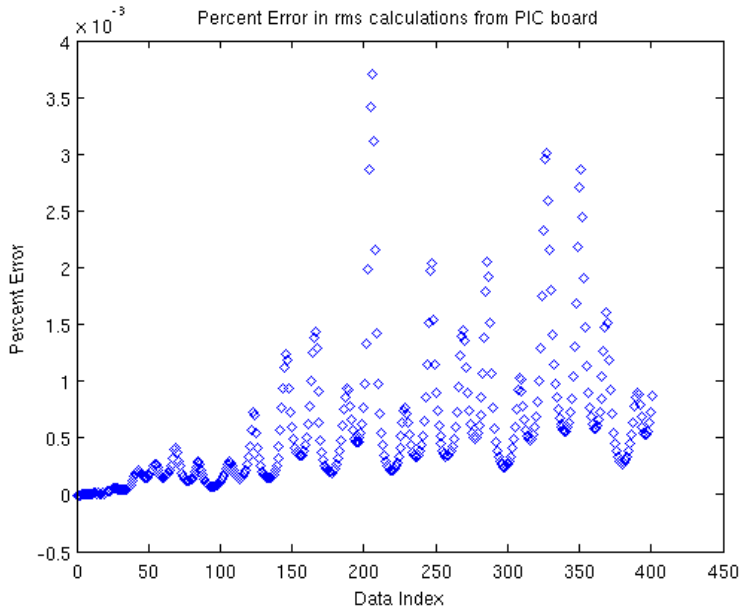

Figure 5: Percent error in PIC18 RMS calculation as compared to MATLAB simulation

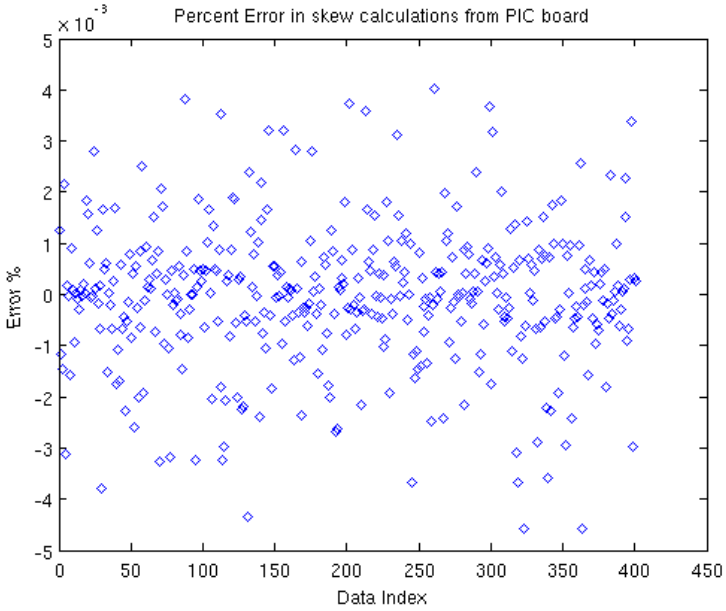

Figure 6: Percent error in PIC18 skew calculation compared to MATLAB simulation

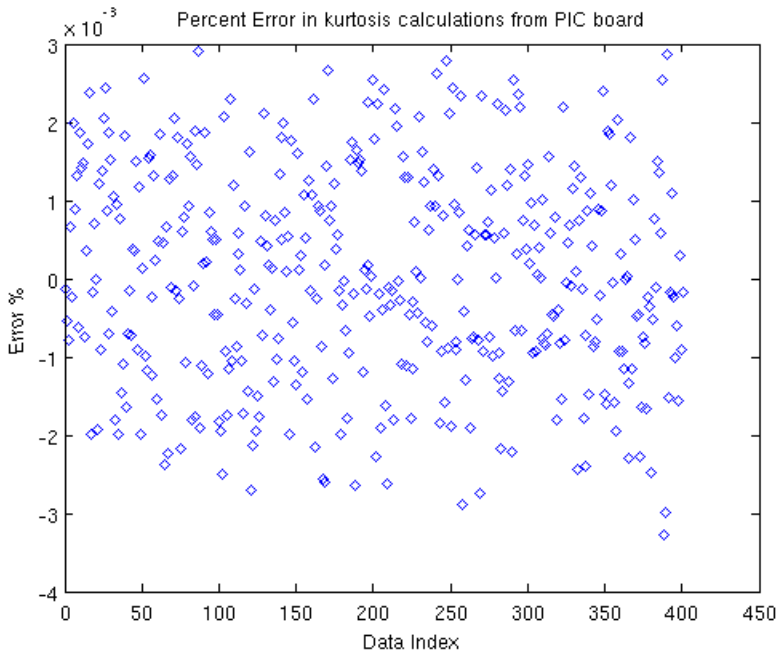

Figure 7: Percent error in PIC18 kurtosis calculation compared to MATLAB simulation

3.1.3 dsPIC Results: Performing RMS calculation on IMU data

To test the dsPIC board, the IMU was attached to the serial port on the dsPICDEM 1.1 development board. After the dsPIC decoded the data, it calculated the RMS values for the three linear accelerations using a ten-sample window. The resulting data was transferred to the computer at 115.2 kbps where a modified VB.NET program was listening. The program was modified to show the RMS values in text fields and superimpose the values on the graph; a screenshot of the program is pictured in Figure 8. The IMU was jostled beginning at approximately packet 9650 and ending at packet 9710. This resulted in spikes in the accelerations along the x, y, and z axes. During this time, the RMS value changed very little. The slight delay in the response of the RMS value is a

result of the sliding window used; since the window moves 10 samples at a time, the impact of the jostling would not show up until after a new set of samples is taken. This test showed proper functionality of the dsPIC board, as well as its ability to perform data processing.

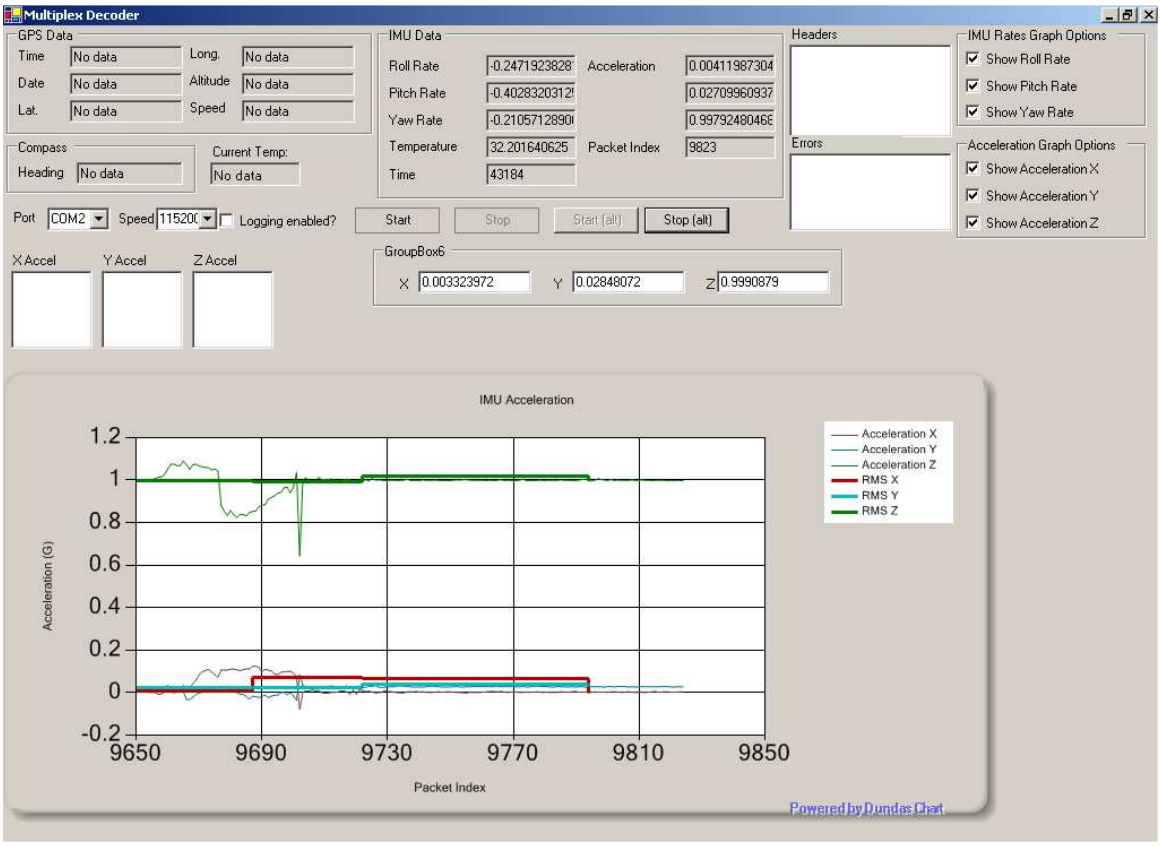

Figure 8: RMS and unprocessed values from dsPIC board connected to the IMU

3.1.4 dsPIC Results: Analog-to-digital converter

To support higher bandwidth data, the analog-to-digital converter (ADC) on the dsPIC was tested. The analog voltage outputs from the accelerometers on-board the IMU were connected to pins AN10-12 on the dsPIC; pin AN13 was connected to ground. The dsPIC has only one ADC, so a multiplexer is used to switch between inputs. Due to the design of the ADC, the multiplexer had to be connected to ground between samples to get rid of excess charge in capacitors located in the ADC. The ADC was configured to be triggered by an overflow in 32-bit timer 2-3, with automatic conversion of the analog data after sampling. The timer was configured to give an effective sampling rate of 199.8Hz, which is approximately the bandwidth required by the IMU. Finally, the ADC interrupt was configured to trigger after every three samples.

Upon receiving an interrupt, the program copied the sample values into a buffer and incremented a counter variable. In the main program, this counter variable was compared with another variable tracking the main program's location in the buffer; if the two counters differed, the main program calculated applied the appropriate compensation to the sampled values and transmitted them to the computer for display.

The dsPIC was successfully able to read data from the IMU through the ADC, with values that were in the expected range. However, there was a decrease in sensitivity, as compared to the digital output of the IMU – it was less sensitive to environmental vibrations as well as vibrations on the same table as the unit. This decrease in sensitivity is most likely attributable to the precision of the ADC, which is only 12 bits, compared to the 16 bit precision of the digital interface of the IMU. Although in this example, the lack

of precision is not significant, other types of sensors may have greater accuracy, and some algorithms may require the greater precision offered through the digital interface. Thus, despite the higher bandwidth offered by the ADC, the lack of precision does not make it feasible for this system.

3.2 Discussion

The system fulfills its purpose of providing a platform for radio-controlled helicopter health monitoring. It is capable of communicating with sensors, performing processing, transmitting the data to a computer, and displaying it in a graphical interface. Testing indicates that the system works properly and can process data in real-time accurately. However, the system does have a few flaws and limitations that will need to be addressed.

The main limitations in the system are a direct result of hardware limitations in the processors chosen for the system. The primary constraint is the amount of RAM; this limits the number of samples available for processing at any particular time. For example, the dsPIC, although it is capable of calculating fast Fourier transforms (FFT), can only fit a 256 point FFT in RAM. Even if only simple statistics are calculated, the amount of RAM limits the number of samples that can be operated on simultaneously; the PIC18, with 3840 bytes of RAM, can maintain 1920 16-bit samples in memory, while the dsPIC, with 8192 bytes of RAM, can hold 4096 16-bit samples in memory. These numbers assume no RAM is used for other operations, however, approximately 25% of the RAM is used solely for buffering data for communications. There is still plenty of memory for

most low-rate applications since even at a sampling rate of 1kHz, over a second of data would fit in memory.

The processors are also somewhat limited in speed, with the PIC18F8670 limited to 10MIPS and the dsPIC30F6014 limited to a maximum of 30MIPS. However, the speed only becomes a limiting factor for real-time signal processing with high bandwidth sensor data. A sample of functions that might be used in signal processing and their estimated runtimes are listed in Table 1. The data were extracted from Microchip's DSP library documentation and only represents the time necessary to run each function. From this table, the dsPIC is capable of performing a 256 point FFT at a rate of approximately 1300Hz, while more complex operations require considerably more time. Although this makes the dsPIC unsuitable for processing high rate data in real-time, a cascaded network of dsPIC boards might be able to process data at a sufficient rate. Normal arithmetic operations only require one or two instruction cycles; both processors should be capable of performing simple statistics, with the dsPIC capable of handling higher data rates.

Table 1: Speed of various math operations on the dsPIC30F6014 @ 30MHz[2]

Operation	*Instruction cycles required*	*Runtime (ms)*
16x16 Matrix Invert	2,521,742	84.06
15x15 Matrix Transpose	~800	0.03
15x15 * 15x15 Matrix Multiply	245,000	8.17
FFT, 256 pt. complex	21,000	0.70
Inverse FFT, 256 pt. complex	23,000	0.76

Another factor reducing the maximum throughput of the system is the serial interface. The serial interface carrying the data out of either the PIC or dsPIC boards is

[2] Data for table derived from [11].

limited to 57.6kbps and 115.2kbps respectively. Following previous examples, assuming a sample requires 16 bits, the maximum data rates are 3600 samples per second and 7200 samples per second. As with the other limitations, this is not a problem for low-bandwidth sensor data, and it may be adequate for higher bandwidth data, assuming processing can be done on-board to reduce the amount of data required to be transmitted.

In addition to the hardware limitations stated above, the VB.NET graphical interface, is limited in the speed it can process and display data. Despite the low data rates used in testing, there was approximately a 0.5s of lag in displaying data on a computer with a 1.8ghz Pentium 4 running Windows 2000. This was subjectively determined by observing the system's response to added vibrations. Initial versions of the program would crash shortly after receiving the initial data packet due to overflowing the communication buffers used in the program. The program was improved so that it would no longer crash; this required switching serial port classes and reducing the update interval of the interface. Some of the lag can be attributed to the use of Microsoft's .NET framework and inefficiencies in the graphing component used to generate the real-time graphs. Despite the lag, the program is effective at conveying the current data from the sensors and works well for testing purposes.

3.3 Conclusion

This system communicates with multiple sensors connected to several microcontrollers, performs processing on the data, and transmits it to a computer where it is finally displayed in real-time. It fulfills all the goals proposed for the system, including

low-cost, low power consumption, and compact size. The operation of the system was verified by calculating simple statistics of data obtained directly from the sensors in real-time. Although all the testing was performed with very low bandwidth data, only several hundred samples per second, the system should scale easily to data rates of 1000 samples per second. The system can be expanded with additional hardware, primarily RAM, to support higher bandwidth data.

There are several additions to the system that would increase its functionality. The next iteration of the system will involve constructing much smaller printed circuit boards for the system instead of the large development boards used in the current system. This will further reduce the space necessary and increase the number of boards that can be mounted on the helicopter. The boards would ideally be small enough to be able to devote an entire board to each sensor – each board would be responsible for translating each sensor's proprietary protocol into a common protocol to be transmitted to other boards for processing and eventual transmission out of the system. These boards could use either a mix of PIC18F8720 and dsPIC30F6014 processors, or just the dsPIC processor. Having several boards would allow additional sensors to be connected to the system for a more complete view of the helicopter's health. Another requested feature is wireless capability. Currently, the system is tethered to a computer though a serial cable. A wireless system could replace the serial cable, enabling the system to be mounted in the RC helicopter and transmit data while in flight. There are many wireless systems available that can connect directly to a serial port; thus, no program changes would be required to interface with the wireless system. Once the system is converted to use

wireless for its connectivity to the computer, actual flight-testing on an RC helicopter can be performed.

Once complete, this system could form the basis for a system for a condition monitoring system for helicopters and other complex mechanical systems that could be much more inexpensive than current systems. Furthermore, the accessibility of the programming environment will promote more experimental testing of algorithms designed to pinpoint faults.

This multi-sensor embedded microcontroller system for condition monitoring of radio-controlled helicopters fulfills all the desired features for this iteration of the system. It is capable of forming the base for a more extensive system that could actually be implemented for condition monitoring of actual helicopters.

References

1. Colucci, Frank. Sensors aboard helicopters can help predict parts failures. National Defense. Arlington: Jan 2005. Vol. 89, Iss. 614; p. 25 (2 pages)

2. PIC18F6520/8520/6620/8620/6720/8720 Data Sheet. Microchip. [Online document]. Available at http://ww1.microchip.com/downloads/en/DeviceDoc/39609b.pdf

3. dsPIC30F6011, dsPIC30F6012, dsPIC30F6013, dsPIC30F6014 Datasheet. Microchip. [Online document]. Available at http://ww1.microchip.com/downloads/en/DeviceDoc/70117e.pdf

4. U.S. Gazetteer. U.S. Census Bureau. [Online database]. Available at http://www.census.gov/cgi-bin/gazetteer?city=&state=&zip=16801

5 Anderson, Peter. DS1820 Digital Thermometer – Calculating an 8-bit CRC Value. [Online document]. June 1998. Available at http://www.phanderson.com/PIC/16C84/crc.html

6 Application Note 27: Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor iButton Products. Dallas Semiconductor. [Online document]. Mar. 29, 2001. Available at http://www.maxim-ic.com/appnotes.cfm/appnote_number/542

7. IMU User's Manual, Models IMU30CC-, IMU400CC-. Crossbow Technology. [Online document]. March 2005. Available at http://www.xbow.com/Support/Support_pdf_files/IMU_Users_Manual.pdf

8. Superstar II User Manual. Novatel Inc. [Online document]. Mar. 2004. Available at http://www.novatel.com/Documents/Manuals/om-20000077.pdf

9. L1 GPS Firmware Reference Manual. Novatel Inc. [Online document]. Mar. 2004. Available at http://www.novatel.com/Documents/Manuals/om-20000086.pdf

10. HMR3100 Digital Compass Solution User's Guide. Honeywell Sensor Products. [Online document]. Feb. 2004. Available at http://www.ssec.honeywell.com/magnetic/datasheets/hmr3100_manual.pdf

11. dsPIC Language Tools Libraries. Microchip. [Online document]. Oct. 26, 2004. Available at http://ww1.microchip.com/downloads/en/DeviceDoc/51456b.pdf

Appendix A

List of Materials

A.1 Hardware

Table 2: Bill of materials

Qty	Item	Part number	Vendor	Unit Cost	Total Cost
1	dsPICDEM 1.1 Demo Board	DM300014	Newark Electronics	299	299
3	RS232 Cable and Power Supply		Newark Electronics	29	87
2	64/80-pin TQFP Demo Board	DM183020	Arrow Electronics	49	98
1	Honeywell digital compass development kit	HMR3100-Demo-232	Honeywell	186	186
1	Novatel Superstar II	SSII-5-STD	NavtechGPS	115	115
1	6DOF IMU	IMU400CC-200	Crossbow Technology	3995	3995
2	Maxim UART controller	MAX3100CPD	Digikey	7	14
2	Maxim RS232 driver	MAX232CPE	Digikey	3.31	6.62
2	Crystal, 3.6864MHz	HC49US3.6864MABJ	Digikey	0.88	1.72
10	0.1μF capacitor	A104K15X7RF5TAA	Digikey	0.35	3.50
4	22pF capacitor	A220J15C0GH5TAA	Digikey	0.09	0.36
2	DIP socket, 14 pin	115-93-314-41-003000	Digikey	1.18	2.36
2	DIP socket, 16 pin	115-93-316-41-003000	Digikey	1.35	2.70
4	DB9 connector, male (plug)	152-309	Mouser	0.99	3.96

A.2 Software and development tools

Table 3: List of software and development tools

Qty.	Item	Part number	Vendor	Unit Cost	Total Cost
1	Microchip MPLAB IDE (7.00)	SW007002	Microchip	0	0
1	Microchip C30 Compiler (1.20)	SW006012	Newark Electronics	895	895
1	CCS PCWH Compiler	PCWH	CCS	425	425
1	Microchip ICD2 Programmer	DV164005	Newark Electronics	159	159
2	CCS ICD-U40 Programmer	ICD-U40	CCS	75	150
1	Microsoft Visual Studio.NET 2003 (Academic)	BBWNSN017	PSU Computer Store	84.62	84.62

Appendix B

Processor Specifications

B.1 PIC18F8720

MICROCHIP

PIC18F6520/8520/6620/8620/6720/8720

64/80-Pin High-Performance, 256 Kbit to 1 Mbit Enhanced Flash Microcontrollers with A/D

High-Performance RISC CPU:

- C compiler optimized architecture/instruction set:
  - Source code compatible with the PIC16 and PIC17 instruction sets
- Linear program memory addressing to 128 Kbytes
- Linear data memory addressing to 3840 bytes
- 1 Kbyte of data EEPROM
- Up to 10 MIPs operation:
  - DC – 40 MHz osc./clock input
  - 4 MHz – 10 MHz osc./clock input with PLL active
- 16-bit wide instructions, 8-bit wide data path
- Priority levels for interrupts
- 31-level, software accessible hardware stack
- 8 x 8 Single Cycle Hardware Multiplier

External Memory Interface (PIC18F8X20 Devices Only):

- Address capability of up to 2 Mbytes
- 16-bit interface

Peripheral Features:

- High current sink/source 25 mA/25 mA
- Four external interrupt pins
- Timer0 module: 8-bit/16-bit timer/counter
- Timer1 module: 16-bit timer/counter
- Timer2 module: 8-bit timer/counter
- Timer3 module: 16-bit timer/counter
- Timer4 module: 8-bit timer/counter
- Secondary oscillator clock option – Timer1/Timer3
- Five Capture/Compare/PWM (CCP) modules:
  - Capture is 16-bit, max. resolution 6.25 ns (TCY/16)
  - Compare is 16-bit, max. resolution 100 ns (TCY)
  - PWM output: PWM resolution is 1 to 10-bit
- Master Synchronous Serial Port (MSSP) module with two modes of operation:
  - 3-wire SPI™ (supports all 4 SPI modes)
  - I^2C™ Master and Slave mode
- Two Addressable USART modules:
  - Supports RS-485 and RS-232
- Parallel Slave Port (PSP) module

Analog Features:

- 10-bit, up to 16-channel Analog-to-Digital Converter (A/D):
  - Conversion available during Sleep
- Programmable 16-level Low-Voltage Detection (LVD) module:
  - Supports interrupt on Low-Voltage Detection
- Programmable Brown-out Reset (PBOR)
- Dual analog comparators:
  - Programmable input/output configuration

Special Microcontroller Features:

- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- 1 second programming time
- Flash/Data EEPROM Retention: > 40 years
- Self-reprogrammable under software control
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own On-Chip RC Oscillator for reliable operation
- Programmable code protection
- Power saving Sleep mode
- Selectable oscillator options including:
  - 4X Phase Lock Loop (of primary oscillator)
  - Secondary Oscillator (32 kHz) clock input
- In-Circuit Serial Programming™ (ICSP™) via two pins
- MPLAB® In-Circuit Debug (ICD) via two pins

CMOS Technology:

- Low-power, high-speed Flash technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Industrial and Extended temperature ranges

Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8-bit/16-bit	Ext Bus	Max Fosc (MHz)
	Bytes	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)				SPI	Master I^2C				
PIC18F6520	32K	16384	2048	1024	52	12	5	Y	Y	2	2/3	N	40
PIC18F6620	64K	32768	3840	1024	52	12	5	Y	Y	2	2/3	N	25
PIC18F6720	128K	65536	3840	1024	52	12	5	Y	Y	2	2/3	N	25
PIC18F8520	32K	16384	2048	1024	68	16	5	Y	Y	2	2/3	Y	40
PIC18F8620	64K	32768	3840	1024	68	16	5	Y	Y	2	2/3	Y	25
PIC18F8720	128K	65536	3840	1024	68	16	5	Y	Y	2	2/3	Y	25

© 2004 Microchip Technology Inc.

Figure 9: PIC18F8720 specifications[3]

[3] Specifications taken from [2]

B.2 dsPIC30F6014

MICROCHIP dsPIC30F6011/6012/6013/6014

dsPIC30F6011/6012/6013/6014 High Performance Digital Signal Controllers

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

High Performance Modified RISC CPU:

- Modified Harvard architecture
- C compiler optimized instruction set architecture
- Flexible addressing modes
- 84 base instructions
- 24-bit wide instructions, 16-bit wide data path
- Up to 144 Kbytes on-chip Flash program space
- Up to 48K instruction words
- Up to 8 Kbytes of on-chip data RAM
- Up to 4 Kbytes of non-volatile data EEPROM
- 16 x 16-bit working register array
- Up to 30 MIPs operation:
  - DC to 40 MHz external clock input
  - 4 MHz-10 MHz oscillator input with PLL active (4x, 8x, 16x)
- Up to 41 interrupt sources:
  - 8 user selectable priority levels
  - 5 external interrupt sources
  - 4 processor traps

DSP Features:

- Dual data fetch
- Modulo and Bit-reversed modes
- Two 40-bit wide accumulators with optional saturation logic
- 17-bit x 17-bit single cycle hardware fractional/ integer multiplier
- All DSP instructions are single cycle
  - Multiply-Accumulate (MAC) operation
- Single cycle ±16 shift

Special Microcontroller Features (Cont.):

- Fail-Safe Clock Monitor operation:
  - Detects clock failure and switches to on-chip low power RC oscillator
- Programmable code protection
- In-Circuit Serial Programming™ (ICSP™)
- Selectable Power Management modes:
  - Sleep, Idle and Alternate Clock modes

Peripheral Features:

- High current sink/source I/O pins: 25 mA/25 mA
- Five 16-bit timers/counters; optionally pair up 16-bit timers into 32-bit timer modules
- 16-bit Capture input functions
- 16-bit Compare/PWM output functions:
- Data Converter Interface (DCI) supports common audio Codec protocols, including I^2S and AC'97
- 3-wire SPI™ modules (supports 4 Frame modes)
- I^2C™ module supports Multi-Master/Slave mode and 7-bit/10-bit addressing
- Two addressable UART modules with FIFO buffers
- Two CAN bus modules compliant with CAN 2.0B standard

Analog Features:

- 12-bit Analog-to-Digital Converter (A/D) with:
  - 100 Ksps conversion rate
  - Up to 16 input channels
  - Conversion available during Sleep and Idle
- Programmable Low Voltage Detection (PLVD)
- Programmable Brown-out Detection and Reset generation

Special Microcontroller Features:

- Enhanced Flash program memory:
  - 10,000 erase/write cycle (min.) for industrial temperature range, 100K (typical)
- Data EEPROM memory:
  - 100,000 erase/write cycle (min.) for industrial temperature range, 1M (typical)
- Self-reprogrammable under software control
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Flexible Watchdog Timer (WDT) with on-chip low power RC oscillator for reliable operation

CMOS Technology:

- Low power, high speed Flash technology
- Wide operating voltage range (2.5V to 5.5V)
- Industrial and Extended temperature ranges
- Low power consumption

dsPIC30F6011/6012/6013/6014 Controller Families

Device	Pins	Program Memory		SRAM Bytes	EEPROM Bytes	Timer 16-bit	Input Cap	Output Comp/Std PWM	Codec Interface	A/D12-bit 100 Ksps	UART	SPI™	I^2C™	CAN
		Bytes	Instructions											
dsPIC30F6011	64	132K	44K	6144	2048	5	8	8	—	16 ch	2	2	1	2
dsPIC30F6012	64	144K	48K	8192	4096	5	8	8	AC'97, I^2S	16 ch	2	2	1	2
dsPIC30F6013	80	132K	44K	6144	2048	5	8	8	—	16 ch	2	2	1	2
dsPIC30F6014	80	144K	48K	8192	4096	5	8	8	AC'97, I^2S	16 ch	2	2	1	2

© 2004 Microchip Technology Inc. Preliminary

Figure 10: dsPIC30F6014 specifications[4]

[4] Specifications taken from [3]

Appendix C

Sensor Communication

C.1 IMU: Crossbow IMU400CC

The Crossbow IMU400CC is a six-degree-of-freedom inertial measurement unit capable of measuring linear acceleration along three orthogonal axes and rotation rates around three orthogonal axes. It communicates using the RS-232 protocol at 38400 bits per second, 8 data bits, no parity, and one stop bit. The IMU can be configured to transmit the voltage measurements from its internal sensors from an analog to digital converter or to perform additional calculations and return the scaled and compensated measurements. For this system, the scaled mode is used. In this mode, the IMU transmits a 18-byte data packet. The packet format is listed in Table 4.

As noted in the packet format listing, the packet begins with a header byte, represented by FF in hexadecimal. Following the header byte are all the measurements by the IMU. Each measurement is two bytes, with the most significant byte transmitted first. All the measurements are signed except for the time and temperature. Finally, the checksum is calculated by taking the sum of all the packet contents except the header and checksum, dividing by 256, and taking the remainder.

The IMU requires initialization before it will send data. This initialization sequence is performed by sending an ‘R’ to the IMU for a reset, followed by a ‘z’ and a number indicating the number of measurements to average over to zero the IMU. Next, a

'c' is transmitted to set the IMU to scaled sensor mode. For the PIC board, to prevent data overflow errors, a 'P' is transmitted to set the IMU to polled mode – in this configuration, the IMU will only send data if data is requested by a 'G'. The dsPIC is capable of handling the IMU continuously without any problems, so a 'C' is sent instead to configure the IMU to send data continuously. This information was taken from Crossbow documentation in [7].

Table 4: IMU Data Packet Format[5]

Byte	Data description
1	Header
2	Roll Rate (MSB)
3	Roll Rate (LSB)
4	Pitch Rate (MSB)
5	Pitch Rate (LSB)
6	Yaw Rate (MSB)
7	Yaw Rate (LSB)
8	Acceleration X (MSB)
9	Acceleration X (LSB)
10	Acceleration Y (MSB)
11	Acceleration Y (LSB)
12	Acceleration Z (MSB)
13	Acceleration Z (LSB)
14	Temperature Voltage (MSB)
15	Temperature Voltage (LSB)
16	Time (MSB)
17	Time (LSB)
18	Checksum

[5] Table based on data in [7]

C.2 GPS: Novatel Superstar II

The Novatel Superstar II GPS receiver communicates using standard RS-232 with the following port settings: 9600 bits per second, no parity, eight data bits, one stop bit, and no handshaking. The receiver transmits the least significant bit first and the most significant bit last, which is opposite from the order used by the other two sensors. Due to a GPS receiver being a complex device with many configuration options, this Novatel receiver can both send and receive several different message types; this system is only concerned with the "Navigation Data (user coordinates)" message which contains the best position and velocity computed by the receiver. The format of this message is summarized in Table 5. Note that much of the data after ground speed is not useful in this application; the program reads the data, but then ignores it for processing and retransmission purposes.

The header is composed of four bytes – the start of header byte (01 in hexadecimal), a message id number, the ones complement of the message id, and finally the message length. The message data listed in Table 5 follows next. Note that many of the measurements are composed of several bytes depending on the desired precision; for example, the longitude and latitude are eight bytes (double data types) to more closely match the precision of the GPS receiver. After the message data is transmitted, a checksum byte follows to ensure the data was transmitted reliably. The checksum is simply the 16-bit sum of all the bytes in the packet treated as unsigned 8-bit numbers, beginning from the start of header byte and ending at the byte preceding the checksum byte. Any overflow is immediately discarded.

The GPS has an optional initialization sequence; the GPS, upon receiving power, immediately begins transmitting data continuously. Both the PIC and dsPIC utilize this method of operation.

This information was taken from Novatel's documentation in [8,9].

Table 5: GPS receiver packet format[6]

Byte Number	Data description
1-4	Header
5	UTC Time, hours
6	UTC Time, minutes
7-14	UTC Time, seconds
15	UTC Date, day
16	UTC Date, month
17-18	UTC Date, year
19-26	Latitude (radians)
27-34	Longitude (radians)
35-38	Altitude (m)
39-42	Ground speed (m/s)
43-46	Track angle
47-50	North velocity
59-62	HFOM
63-66	VFOM
67-68	HDOP
69-70	VDOP
71-73	NAV mode, configuration, status
74	Elapsed time since power up (hours)
75	Reserved
76-77	Checksum

[6] Table based on data in [9].

C.3 Compass: Honeywell HMR3100

The Honeywell HMR3100 is a digital compass with an RS-232 interface. The default data bit format is 9600 bps, eight data bits, no parity, and one stop bit. The compass has a relatively simple protocol, depicted in Table 6 . The header is either 80 or 81 in hexadecimal; 81 indicates possible magnetic interference. This system ignores this indicator, since the data returned is still correct. The next two bytes are the heading as an unsigned 16-bit integer; the physical heading is determined by dividing the number by two. The compass requires use of the RTS serial line for complete configuration; however, the additional ports on the PIC boards do not have this line connected to lower the number of additional ICs needed. Thus, although the compass can be configured to send data continuously by holding the RTS line low, the system polls the compass for data by transmitting a 1. This information was taken from Honeywell's documentation in [10].

Table 6: Compass packet format[7]

Byte	Data description
1	Header
2	Heading (MSB)
3	Heading (LSB)

[7] Table based on data in [10].

Appendix D

Dallas Semiconductor One-wire Cyclic Redundancy Check

An effective error detection scheme for serial data is the cyclic redundancy check (CRC). There are many different CRC algorithms; for this system, the Dallas One-wire (DOW) CRC is used. This CRC is capable of detecting the following errors:

1. Any odd number of errors anywhere within a 64-bit number
2. All double-bit errors anywhere within the 64-bit number
3. Any cluster of errors that can be contained within an 8-bit "window" (1-8 bits incorrect)

The algorithm is best explained by modeling an 8-bit shift register. For each data bit, the bit is exclusive OR'd with the least significant bit of the shift register. This result is called the "feedback bit." The feedback bit is then exclusive OR'd with bits 4 and 3 in the shift register, the shift register is shifted one bit to the right, and the feedback bit is inserted into the most significant position. This might be more easily understandable if it is actually written out (example reproduced from [5]).

If the shift register was originally;

```
D7 D6 D5 D4  D3 D2 D1 D0
```

The new value is

```
FB D7 D6 D5  D4^FB D3^FB D2 D1
```

Where;

```
FB = D0 ^ data_bit
```

To determine if the received data is free of errors, this process is repeated on the receiver, and the calculated checksum and received checksums are compared. If they differ, a transmission error has occurred. Since this algorithm only utilizes bit shifting and bitwise operators, it executes quickly on the processors [5, 6].

Appendix E

Schematics

E.1 Additional RS-232 ports

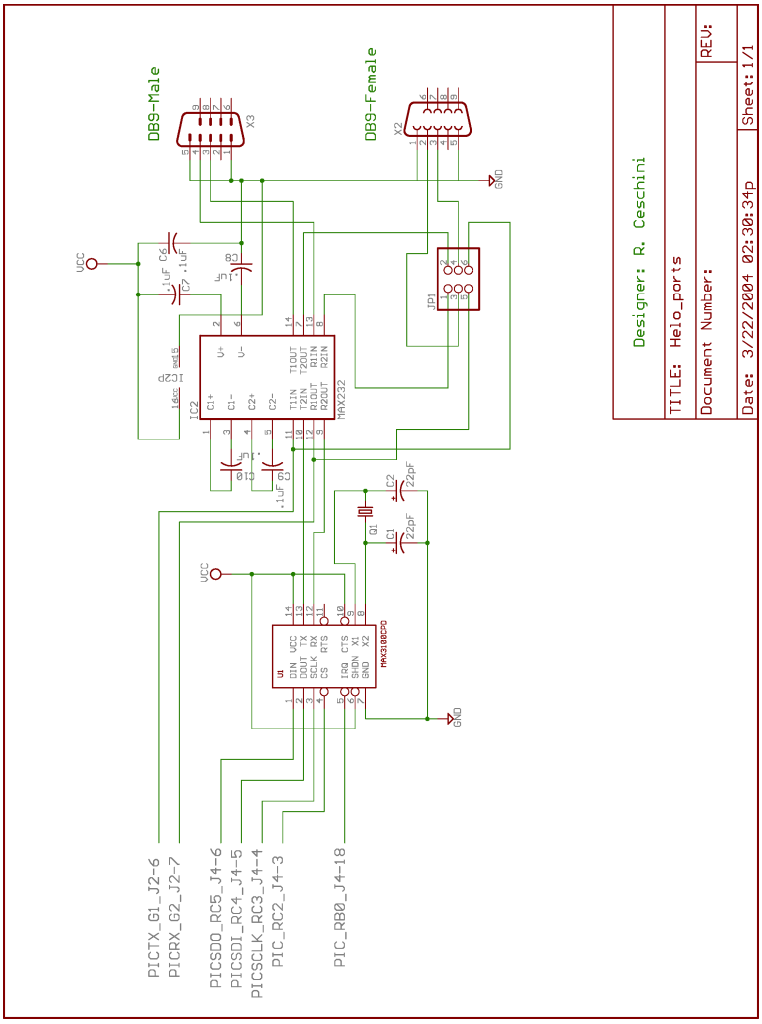

Figure 11: Schematic of additional RS-232 ports[8]

[8] Schematic design by Robert Ceschini, Applied Research Lab at Pennsylvania State University

Appendix F

Academic Vita of Artit Jirapatnakul

Artit Jirapatnakul
artitj@gmail.com

Education

Pennsylvania State University 2001 – 2005
Bachelor of Science, Electrical Engineering
Honors in Electrical Engineering

Thesis Title: A Multi-sensor Embedded Microcontroller System for Condition Monitoring of RC Helicopters
Thesis Supervisor: Dr. Amulya K. Garga

Work Experience

Computer and Network Support Assistant Oct. 2002 – May 2005
Intercollegiate Athletics, Pennsylvania State University
Supervisor: Alan Claver
Performed support for all Intercollegiate Athletics staff and computer equipment

Web Application Programmer June 2003 – May 2005
Center for Language Acquisition, Pennsylvania State University
Supervisor: Scott Payne
Programmed web-based data collection application using PHP and MySQL.

Teaching Assistant June 2003 – Aug. 2003
Pennsylvania Governor's School for Information Technology at Drexel University
Supervisor: Brian Finnegan
Taught eighty high school students programming and advanced topics in IT.

Awards:

Lockheed Martin Engineering Scholar

Professional Memberships:

Institute of Electrical and Electronics Engineers

Presentations:

Participant in Undergraduate Research Exhibition
Pennsylvania State University, April 2005